Personal SE

C Struct & Typedef
C Structs

- A *struct* is a way of grouping named, heterogeneous data elements that represent a coherent concept.
A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.

Example:

```c
#define MAXNAME (20)

struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```
Question: What is an object with no methods and only instance variables public?

Answer: A struct! (well, sort of).

A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.

Example:

```
#define MAXNAME (20)

struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```
C Structs

- Question: What is an object with no methods and only instance variables public?
- Answer: A struct! (well, sort of).
- A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.
- Example:

```c
#define MAXNAME (20)
struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

heterogeneous - the fields have different types
Question: What is an object with no methods and only instance variables public?

Answer: A struct! (well, sort of).

A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.

Example:

```c
#define MAXNAME (20)

struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

coherent concept - the information recorded for a person.
Using Structs

• Declaration:

```c
struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

• Definitions:

```c
struct person mike, pete, chris ;
```

• Assignment / field references ('dot' notation):

```c
mike = pete ;
pete.age = chris.age + 3
```
Using Structs

• Note: Space allocated for the whole struct at definition.
• Struct arguments are passed by value (i.e., copying)

```c
void give_raise(struct person p, double pct) {
    p.income *= (1 + pct/100);
    return ;
}
give_raise(mike, 10.0);
```

```c
struct person give_raise(struct person p, double pct) {
    p.income *= (1 + pct/100);
    return p;
}
mike = give_raise(mike, 10.0);
```
Symbolic Type Names - typedef

- Suppose we have a pricing system that prices goods by weight.
  - Weight is in pounds, and is a double precision number.
  - Price is in dollars, and is a double precision number.
  - Goal: Clearly distinguish weight variables from price variables.
Symbolic Type Names - typedef

• Suppose we have a pricing system that prices goods by weight.
  – Weight is in pounds, and is a double precision number.
  – Price is in dollars, and is a double precision number.
  – Goal: Clearly distinguish weight variables from price variables.

• Typedef to the rescue:
  – typedef declaration ; Creates a new "type" with the variable slot in the declaration.
Symbolic Type Names - typedef

- Suppose we have a pricing system that prices goods by weight.
  - Weight is in pounds, and is a double precision number.
  - Price is in dollars, and is a double precision number.
  - Goal: Clearly distinguish weight variables from price variables.

- **Typedef to the rescue:**
  - typedef *declaration* ; Creates a new "type" with the variable slot in the declaration.

- **Examples:**
  ```c
  typedef double price ;          // alias for double to declare price variables
  typedef double weight ;        // alias for double to declare weight variables
  price p ;                     // double precision value that's a price
  weight lbs ;                  // double precision value that's a weight
  ```
typedef In Practice

- Symbolic names for array types

```c
#define MAXSTR (100)

typedef char long_string[MAXSTR+1];

long_string line;
long_string buffer;
```
typedef In Practice

• Symbolic names for array types

    #define MAXSTR (100)

typedef char long_string[MAXSTR+1] ;

    long_string  line ;
    long_string  buffer ;

• Shorter name for struct types:

typedef struct {
    long_string  label ;  // name for the point
    double  x ;  // xcoordinate
    double  y ;  // ycoordinate
} point ;

    point  origin ;
    point  focus ;